SYSTEM AND METHOD FOR COMPILING AN EXTENSIBLE MARKUP LANGUAGE BASED QUERY

FIELD OF THE INVENTION

[0001]   The present invention relates to the field of data retrieval and manipulation, and, more specifically, to compiling an extensible markup language (XML) based query.

BACKGROUND OF THE INVENTION

[0002]   In conventional databases, an incoming query is typically received and manipulated by a database front end prior to being submitted to a query processor for optimization and execution. Generally, the front end uses the incoming query to generate a query plan for executing the query at the query processor. Generating an execution plan is a multi-stage process. For example, SQL Server™ from Microsoft Corp. of Redmond, Washington, is a relational database in which an execution plan may be generated as follows. First, the query may be parsed to yield an abstract syntax tree. The abstract syntax tree may then be transformed into a unified tree structure in which nodes represent abstract operations to be performed on the query. An algebrizer may then be employed to convert the unified tree operations into relational algebraic expressions in a log-op tree. The log-op tree is the resulting query plan which is submitted to the query processor for optimization and execution.

[0003]   A recent development with respect to databases is that, in addition to supporting traditional relational data, the databases also support extensible markup language (XML) data. For example, SQL Server enables data to be defined using an XML data type. Columns with such an XML data type can be created in a data table, and XML variables and parameters can be declared. Such XML data can be searched, retrieved, and updated. Specifically, a query may include an XML expression written in an XML based query language such as XQuery and XSLT.

[0004]   A limitation of conventional databases is that the existing method of generating a query plan set forth above is inadequate for queries that

include such an XML expression. Specifically, the existing relational operations are inadequate for XML expressions that include document order and hierarchical ordering of data. Furthermore, existing algebrizers are inadequate to convert XML operations into relational algebra operations. Furthermore, existing relational algebra operations are inadequate to describe XML operations.

[0005]   Thus, there is a need in the art for systems and methods for compiling an XML based query. The present invention satisfies these needs.

SUMMARY OF THE INVENTION

[0006]   The present invention is directed to systems and methods for compiling an extensible markup language (XML) based query. According to the invention, a query including an XML expression is received by a database engine front end. The query is parsed by a relational query parser and embedded XML expressions are forwarded to an XML parser. The XML expression is parsed by the XML parser to yield an abstract syntax tree. The abstract syntax tree is then forwarded to an XML operation generator, which transforms the abstract syntax tree into a unified tree including XML algebra operations. The unified tree is then forwarded to an XML algebrizer, which converts the unified tree into a relational tree including enhanced relational algebra operations. The relational tree is grafted into a log-op tree being produced for the enclosed relational query. The log-op tree results in a query plan that is forwarded to a query processor for optimization and execution of the query at the query processor.

[0007]   According to an aspect of the invention, the enhanced relational algebra operations include nested table abstraction operations. Such operations may include, for example, a row nesting operation, a nested table expansion, a nested row expansion, and a nested row descendant expansion.

[0008]   Additional features and advantages of the invention will be made apparent from the following detailed description of illustrative embodiments that proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009]   The illustrative embodiments will be better understood after reading the following detailed description with reference to the appended drawings, in which:

[0010]   Fig. 1 is a block diagram of an exemplary prior art query processing environment;

[0011]   Fig. 2 is a block diagram of an exemplary prior art database front end;

[0012]   Fig. 3 is a block diagram of an exemplary database in accordance with the present invention;

[0013]   Fig. 4 shows an exemplary XML schema in accordance with the present invention;

[0014]   Fig. 5 shows exemplary instances of an XML document conforming to an XML schema in accordance with the present invention;

[0015]   Fig. 6 shows a block diagram of an exemplary database front end in accordance with the present invention;

[0016]   Fig. 7 shows an exemplary abstract syntax tree in accordance with the present invention

[0017]   Fig. 8 shows an exemplary unified tree in accordance with the present invention;

[0018]   Figs. 9a and 9b show an exemplary relational tree in accordance with the present invention;

[0019]   Fig. 10 shows an exemplary interface class of algebra operations in accordance with the present invention;

[0020]   Fig. 11 shows an exemplary algebrization of a LET expression in accordance with the present invention;

[0021]   Fig. 12 shows another exemplary algebrization of a LET expression in accordance with the present invention;

[0022]   Fig. 13 is a block diagram representing an exemplary network environment having a variety of computing devices in which the present invention may be implemented; and

[0023]   Fig. 14 is a block diagram representing an exemplary computing device in which the present invention may be implemented.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

[0024]   The subject matter of the present invention is described with specificity to meet statutory requirements.   However, the description itself is not intended to limit the scope of this patent.   Rather, the inventors have contemplated that the claimed subject matter might also be embodied in other ways, to include different steps or elements similar to the ones described in this document, in conjunction with other present or future technologies.

1.   Exemplary Query Processing Environment

[0025]   An exemplary prior art query processing environment is shown in Fig. 1. Generally, client application 220 submits a query 305 to database 210. The query is then processed at database 210, and results are returned to client application 220.

[0026]   Database 210 includes database front end 310, query processor 320, and storage processor 330.   Database front end 310 receives query 305 from client application 220 and validates query 305.   For example, front end 310 may validate that a table referenced by query 305 is a valid table that is present in database 210.   Front end 310 may also verify that query 305 has been submitted by an authorized user.   Front end 310 also generates a query plan 315 for executing the query 305, which will be discussed in detail below with reference to Fig. 2.

[0027]   Query processor 320 receives query plan 315 and generates an execution plan based on query plan 315.   Query processor 320 may perform an optimization on query plan 315 to determine the optimal manner in which to execute query plan 315.

[0028]   Storage Processor 330 maintains data in a storage medium. Storage processor 330 also maintains a database schema, which is a hierarchical organization of the data. Essentially, then, storage processor 330 contains the functionality to access the data of database 210 based on a received execution plan from query processor 320.

[0029]   An exemplary prior art database front end 310 is shown in Fig. 2. Generally, front end 310 receives query 305 and generates a corresponding query plan 315, which is submitted to query processor 320 for optimization and execution. Relational parser 422 parses query 305 to yield an abstract syntax tree 325a. Relational operation generator 424 than transforms the abstract syntax tree 325a into a unified tree structure 335a. The nodes of unified tree structure 335a represent abstract relational operations to be performed for query 305. Algebrizer 426 converts the relational operations in the unified tree 335a to relational algebraic expressions in a log-op tree 345. The log-op tree 345 is the resulting query plan 315, which is submitted to query processor 320.

2.   Systems and Methods for Compiling an Extensible Markup Language Based Query

[0030]   As set forth above, the present invention is directed to compiling an extensible markup language (XML) based query. The present invention may be implemented in connection with a database that enables data to be defined using an XML data type, such as database 210 of Fig. 3. Database 210 includes a data table 230 with columns 232a and 232b. Column 232a is a relational column that is the primary key column of table 230 including a unique identifier for each instance of column 232b. As should be appreciated, table 230 may include other relational columns (not shown) in addition to column 232a. Column 232b is an XML column that includes XML data organized according to a hierarchical manner which may include elements, attributes, and text. Table 230 may also include additional XML columns (not shown).

[0031]   Referring now to Fig. 4, exemplary XML schema 400 shows a hierarchy for "People" data. For each person that is an instance of "People" schema

400, data may be entered for a corresponding name, email address, phone, address, homepage, credit card, and profile and so on as permitted by the XML schema.

[0032]   Referring now to Fig. 5, document 500 includes data for two instances of "People" schema 400. Each such instance has a corresponding unique identifier that is a value for ID column 232a in Fig. 3.

[0033]   The XML data in XML column 232b may be searched, retrieved, and updated by submitting to database 210 a query 305 that includes an XML expression. The XML expression may be written in XML query language such as, for example, XQuery, XSLT, or an extension of the SQL language. An exemplary XML expression for retrieving data from "People" data table 230 including XML column 232b with "people" schema 400 is shown below:

```
select p_xmlperson.query('
     declare namespace p="people-uri"
     /p:person[p:profile/p:age>21]/p:name')
from people
```

[0034]   This query includes one path expression, namely: /p:person[p:profile/p:age>21]/p:name, which contains the paths   p:person, p:person/p:profile/p:age, and p:person/p:name. To evaluate the expression, "People" table 230 is filtered on XML column 232b with an age predicate (age > 21). The name of each person that satisfies the age predicate is then retrieved.

[0035]   Referring now to Fig. 6, query 305 including XML expression 415 may be received and manipulated by front end 310 to generate a query plan 315 for executing query 305. In addition to parsing query 305 to yield a relational abstract syntax tree 325a, relational parser 422 passes XML expression 415 from query 305 to XML parser 432. XML parser 432 than parses XML expression 415 to generate an XML abstract syntax tree 325b.

[0036]   Referring now to Fig. 7, exemplary XML abstract syntax tree 325b is a bound abstract syntax tree that corresponds to exemplary XML expression 415 above. The process of generating abstract syntax tree 325b is well known in the art and will not be described in detail herein.

[0037]   XML operation generator 434 transforms the XML abstract syntax tree 325b into an XML unified tree structure 335b. Unlike relational operation generator 424, XML operation generator 434 employs a set of XML operations corresponding to XML algebraic expressions.

[0038]   XML operation generator 434 recursively traverses abstract syntax tree 325b. For each node of abstract syntax tree 325b, a corresponding unified sub-tree is generated. Each such unified sub-tree includes a number of XML algebra operations required to perform the expression in its associated abstract syntax tree node. The unified sub-trees are then inserted into unified tree 335b.

[0039]   Referring now to Fig. 8, exemplary unified tree structure 335b corresponds to the exemplary XML expression shown above. Tree structure 335b includes a number of XML operations, which are discussed in detail within the "XML Operation Generation" section below.

[0040]   XML Algebrizer 436 converts the XML operations within unified tree structure 335b into a relational log-op tree 345 that includes enhanced relational operations. XML Algebrizer 436 recursively traverses unified tree 335b. For each node of unified tree 335b, a corresponding relational sub-tree is generated. Each such relational sub-tree includes a number of enhanced relational algebra operations required to perform the XML algebra operation in its associated unified tree node. The relational sub-trees are then inserted into relational tree 345.

[0041]   Referring now to Figs. 9a and 9b, exemplary relational log-op tree 345 corresponds to the exemplary XML expression shown above. Tree structure 345 includes a number of enhanced relational operations, which are discussed in detail within the "XML Algebrization" section below.

a.   **XML Operation Generation**

[0042]   XML operation generator 434 transforms the XML abstract syntax tree 325b into an XML unified tree structure 335b including XML algebra operations. Referring now to Fig. 10, the set of operations "COpArg" 1000 includes a set of relational operations 1005 employed by relational algebra generator 424. "CXMLOpArg" 1010d is added to "COpArg" 1000. "CXMLOpArg" 1010d is the set

of XML operations employed by XML operation generator 434. "CXMLOpArg" 1010d includes a number of operations, which are discussed in detail below. Each such operation may accept input such as, for example, an ordered XML node list, an unordered XML node set, a Boolean condition, an ordinal condition, a node list condition, and other scalar input.

[0043]   CXmlOp_Follow is an operation that takes an XML node list as input, and returns a following XML node list as output. The types of following node lists which are output may be, for example: children ("/"), which are followed from a parent to the children; descendants ("//"), which are followed from an ancestor to the descendants; a parent (".."), which is followed from a child to the parent; a self node ("."); and an identifier reference (IDREF), ("->"), which is a reference to a corresponding node.

[0044]   When a child or a descendant follow is performed, a "target" may be set. The target restricts the output list of XML nodes. For example, "a/b" sets the target as "b", so that the output list is restricted to sub-elements with name "b". A target can be elements, attributes, or PCDATA. When an IDREF follow is performed, the source node already contains the name of the referred node. Accordingly, the target restriction is only used for re-validation of an IDREF follow.

[0045]   In addition to a target, a follow operation may include a flag that provides detail as to how the follow is to be applied. Such flags may include, for example, an all elements flag and a relative flag. An all elements flag indicates that the target of the follow is all elements. The all elements flag does not require string comparison. A relative flag may be employed when a follow does not have a specified input such as, for example, in the case of predicates in XPath expressions. For example, in a[b = "c"], the predicate operator takes "a" as one input, and "follow = 'b'" as another input for the condition. The follow operator sets the target as "b", but the target is not bound to any input until the predicate operation is evaluated. The actual input is the left child of its nearest predicate above it in the unified tree. If that child is also a relative follow operator, then the next child above it is evaluated until an XML node list is found. Thus, a predicate defines the context for relative follow operators below it.

[0046]   The relative flag may also be employed to express a nested follow that is not to be evaluated from left to right. For example, "a/b/c" is normally expressed as "(a/b)/c". However, if "a/b/c" is specified as "a/(b/c)", then "b/c" is evaluated first, and, since "b" is not bound, it is expressed as a relative follow.

[0047]   The list of output nodes is ordered according to the order of input nodes. Output nodes that are generated from different input nodes are ordered in the same order of input nodes, while output nodes that are generated from the same input node are ordered in the global document order.

[0048]   XmlOp_Follow is an operation that represents XPath axes. The input of this operator is one node, and the output is a list of nodes following the axis. The follow types may include, for example, child, self, descendant, descendant-or-self, parent, ancestor, ancestor-or-self, following-sibling, preceding-sibling, following, preceding, and IDREF.

[0049]   CXmlOp_Select is an operation that takes a list of ordered XML nodes as a left child and a condition as right child, and returns the qualifying input nodes based on the given condition in their input order. The "select" operator can have three types of conditions: Boolean, ordinal or XML nodes.

[0050]   The Boolean condition is usually expressed as a comparison of an XML node and some constant. The XML node can be the list output of a relative "follow" operation, meaning that the XML node is evaluated based on the input XML node of the "select" operation such as, for example, the list of 'a' nodes in "/a[b = 3]". The XML node can also be evaluated regardless of the current node such as, for example, "/a[/b = 6]". The condition means that the current XML node "a" on which the "select" operation is working must have at least one XML node "b" with which the given relationship is present. The given condition must also be satisfied.

[0051]   The ordinal condition can be either an integer or a list of integers such as, for example, "/a[2]". The condition means that the XML nodes "a" corresponding to the "select" operation must be in one of the given positions among its siblings (i.e. position 2).

[0052]   This XML node list implies checking for existence, which means that the current XML node on which the "select" operation is working must

have at least one XML node with which the given relationship is present. An example is the XPath expression "/a[b]".

[0053]   CXmlOp_Get is an operation that is used to read an XML source. CXmlOp_Construct creates a new XML node.

[0054]   CXml_OpConstruct is an operation that can create all types of XML nodes, namely: elements, attributes, processing instructions, comments, and PCDATA. For an element, the operator takes input, which gives the sub-nodes (attributes and/or children) and indicates the order among the attributes. For an attribute, the operator takes a constant (integer, float, string, etc.) as the input, and creates an attribute with the value as the input constant as the value. For PCDATA, the operator takes a string as the input, and creates a PCDATA element.

[0055]   CXmlOp_Concat is an operation that takes two XML node lists as input, concatenates the second list to the end of the first list, and returns the new list. CXmlOp_Union takes two unordered XML node sets as input, and returns one XML node set which is the union of the two input sets. The union operation is based on node identity.

[0056]   CXmlOp_Except is an operation that takes two unordered XML node sets as input, and returns one XML node set, in which each node exists in the first set but doesn't exist in the second set.

[0057]   CXmlOp_Intersect is an operation that takes two unordered XML node sets as input, and returns one XML node set, in which each node exists in both of the input node sets.

[0058]   CXmlOp_Filter is an operation that takes two XML node lists as input, and returns copies of some of the nodes in the forest represented by the first node list, while preserving their hierarchic and sequential relationships. The nodes that are copied into the result are those nodes that are present at any level in the first operand and are also top-level nodes in the forest represented by the second node list.

[0059]   CXmlOp_If is an operation that takes two XML node lists and a condition as input, and, if the condition is satisfied, returns the first node list. If the condition is not satisfied, the operation returns the second node list.

[0060]   CXmlOp_OrderBy is an operation that takes two XML node lists as input. One node list is the list of nodes to be sorted. The other node list includes only one node, which is evaluated within the context of the individual nodes to be sorted. The sorting can be either ascending or descending.

[0061]   CXmlOp_Before is an operation that is used to search for information by ordinal position. It operates on two lists of nodes and returns those nodes in the first list that occur before at least one node of the second list in document order.

[0062]   CXmlOp_After is an operation that is used to search for information by ordinal position. It operates on two lists of nodes and returns those nodes in the first list that occur after at least one node of the second list in document order.

[0063]   CXmlOp_Var is an operation that is used to refer to a declared node variable. CXmlOp_Var has no input. It has a "name" property which is that of the node variable.   CXmlOp_Var is normally used as the right child of CXmlOp_Apply.

[0064]   CXmlOp_Apply is an operation that takes two XML node lists as input, and returns one node list. It has an "apply name" property whose value is the name of the variable bound by a corresponding FOR clause in XQuery. The variable is bound to each of the nodes in a first node list. A second node list may contain references to this variable.

[0065]   CXmlOP_Insert is an operation that takes two inputs. The first input is an XML node list, which has only one node. The second input can be either an ordered XML node list or an unordered XML node set. The operation inserts a copy of the node in the first list to some position related to the node in the second list or set. The insertion can have three types: INTO, BEFORE, or AFTER. The insertion doesn't have output. For an INTO type, a copy of the node in the first list is inserted as a direct descendent for each of the nodes N of the second list or set. If there are already children for such a node N, then the inserted node is appended to the existing list of children if they are ordered, or otherwise inserted into the unordered children collection of N. For BEFORE or AFTER types, a copy of the node in the first list is

inserted directly after or before each of the nodes of the second list. The second input must be an ordered XML node list.

[0066]   CXmlOp_Delete is an operation that takes an XML node list as input, and deletes all the nodes in the list.

[0067]   CXmlOp_Replace is an operation that takes two XML node lists as input. Each node in the first list is replaced with a copy of the node in the second list. The replaced node's position is preserved. As part of the replacement, the old node including its subtrees is deleted. If the second list results in an empty sequence, then the nodes in the first list are deleted. If the second list results in more than one node, then an error is returned.

[0068]   XmlScaOp_Accessor is an operation that has no input, meaning that it refers to the context node, or current node which can be determined. The operator returns the scalar value of specific properties of XML nodes. Such properties may include, for example, name, nodetype (element, attribute, text, comment, processing instruction), position, and data.

[0069]   XmlScaOp_Function is an operation that represents a built in function that returns a scalar. The inputs are the parameters of the function and the output is the result of the function as a scalar.

[0070]   XmlOp_Function is an operation that represents a built in function that returns XML nodes. The inputs are the parameters of the function and the output is the result of the function as a list of nodes. For XPath expressions, a function NameTypeCheck may be defined. The function has 3 inputs: a list of nodes, a string (name), or an integer (type). The output is a list of nodes that have the requested name and type.

[0071]   XmlOp_XmlNode is an operation that keeps the context node in addition to a bound variable.

b.   XML Algebrization

[0072]   XML Algebrizer 436 converts the XML operations within unified tree structure 335b into a relational log-op tree 345. Conventional relational algebra operations are inadequate to describe XML operations which access

hierarchical data. Accordingly, XML algebrization in accordance with the present invention converts XML operations into enhanced relational algebra operations.

[0073]   Enhanced relational algebra operations in accordance with the present invention may include one or more nested table abstraction operations. Such nested table abstractions may include, for example, a GbAgg operation that nests a set of rows into an existing row, thereby creating a parent / child relationship between the existing row and the nested rows; an UNNEST operation that expands a nested table; an UNNEST* operation that expands all levels of nesting for an existing nested row; and a FLATTEN operation that expands all descendants of an existing nested row.

[0074]   XML algebrization is described in detail below with respect to an XML expression in the XQuery language. However, an XML expression in accordance with the present invention may also be submitted in other XML query languages such as, for example, XSLT and an extension of the SQL language. XML algebrization of an XQuery expression in accordance with the present invention may be divided into algebrization of XPath expressions, XQuery language constructs, XQuery built in functions, and data modification constructs.

i. XPath Expressions

[0075]   An exemplary set of enhanced relational algebra mappings for XPath expressions is shown in Table 1:

Table 1

Path expression	XML Algebra Operation	Enhanced Relational Mapping
/ (Children)	CXMLOp_Follow	UNNEST
// (Descendants)	CXMLOp_Follow	UNNEST* (recursive UNNEST)
[Predicate]	CXMLOp_Select	EXISTS or SELECT
[Ordinal]	CXMLOp_Select	RANK (adds a ranking column that refers to existing position)
=> (IDREF)	CXMLOp_Follow	JOIN

[0076]   The slash operator "/", yielding the children of a node, maps directly to an UNNEST operation. Predicate evaluation is performed within the WHERE clause of a SELECT or by checking whether a row exists that satisfies the

predicate. The Ordinal operation adds a ranking column for ordering.   Following an
IDREF is converted into a JOIN between origin and destination XML nodes.

[0077]   An exemplary enhanced relational mapping of /a//b is:

SELECT b* FROM UNNEST* (a, b).

[0078]   An exemplary enhanced relational mapping of /x/a
[P=5], where P is an element, is:

SELECT a.* FROM UNNEST (x, a) as $b,
WHERE EXISTS (SELECT * FROM UNNEST ($b, P) WHERE P.Value = 5)

ii. XQuery Language Constructs

a. if/then/else

[0079]   An exemplary set of enhanced relational algebra
mappings for XQuery language constructs is shown in Table 2:

Table 2

XQuery expression	*XML Algebra Operation*	*Enhanced Relational Mapping*
WHERE	CXMLOp_Select	SELECT
	Assemble elements	FLATTEN

if test_expr
then expr1
else expr2
		CLogOp_Union(all)
		    CLogOp_Select
		        expr1
		        CScaOp_Exists
		            test_expr
		    CLogOp_Select
		        expr2
		        CScaOp_NotExists
		            test_expr

Some $v in e1 satisfies e2		CScaOp_Exists
		    CLogOp_Select
		        e1
		        e2

Every $v in e1 satisfies e2		CScaOp_NotExists
		    CLogOp_Select
		        e1
		        not(e2)

Construct	CXMLOp_Construct	PROJECT and UNION_ALL

Concat (comma)	CXMLOp_Concat	UNION_ALL
FOR	CXMLOp_Apply	Variable binding (FROM clause) or APPLY
FOR (only Path expression in binding)	CXMLOp_Apply	APPLY
FOR (Path Construct / Concat in binding)	CXMLOp_Apply	APPLY, ClogOp_Spool and CLogOp_Sequence
LET (Path Construct in binding)	CXMLOp_Let	CLogOp_Spool and CLogOp_Sequence (common subexpression spool)

[0080]   With respect to the if/then/else expressions, a "Switch Union" operation may be employed to simplify the mapping. The Switch Union operation chooses one of many children in accordance with a scalar.

[0081]   With respect to the LET expression, LET is best understood as a sub-expression such as shown below:

LET $var = expression1($v1, $v2, ...)
RETURN expression2($var, $v1, $v2, ...)

Expression1 may be effectively assigned to $var, and expression2 can refer to this variable any number of times. In addition, both expressions can depend on variables defined by outer expressions.

[0082]   Two exemplary implementations for converting a LET expression are described below. The implementation plans depend on a GbAgg(nest)/unnest operation and to a lesser extent on exposing the Common Subexpression Spool (CLogOp_Spool) as an input operator.

[0083]   The first exemplary implementation involves the use of simple nested table algebra. It leaves the choice of whether to re-execute or spool nested sub-expressions to query processor 320, which is appropriate given that this is a cost-based decision. An exemplary enhanced relational mapping for an exemplary LET expression under the first implementation is shown in Fig. 11. The LET expression is of the following form:

LET $v = Expr1

RETURN Expr2

[0084]   Steps 1110 – 1118 show that, as part of "Expr1", an item identifier ("ItemId") is manufactured and used by nest as the grouping column. The mapping shown in Fig. 11 assumes that "Expr2" refers to $v in two places.

[0085]   The second exemplary implementation is based on exposing CLogOp_Spool to XML algebrizer 436. An exemplary enhanced relational mapping for the exemplary LET expression above under the second implementation is shown in Fig. 12. Steps 1210-1214 show that an "ItemId" is employed to distinguish multiple different groups of XML nodes flowing into the same spool.

[0086]   The conversion of the LET expression may be applied in situations other than user specified queries. For example, arbitrary expressions in FOR-bindings and ORDER-BY expressions are query transforms that enable implementations for other XQuery constructs. The two exemplary queries below illustrate how arbitrary expressions in a FOR-binding may involve the LET expression. The two exemplary queries are semantically equivalent, but suggest different execution plans:

```
FOR $b in expr1
   RETURN expr2($b)

LET $b1 = expr1
   FOR $b in $b1
      RETURN expr2($b)
```

The second version can be mapped even if expr1 is not subject to reference semantics such as, for example, in the case of path expressions. The second query is advantageous because it isolates the copy via the LET expression and then has the FOR operation refer to this copy.

[0087]   The Order-by expression can be converted to reference semantics in a similar fashion as shown below:

```
<a><b> { /x/c[2] } </b><b> { /y/c[3] } </b></a> ORDER-BY a/b/c
```

to

LET $e = <a> { /x/c[2] } { /y/c[3] }
$e ORDER BY a/b/c

As shown in the examples above, conversion of the LET expression may be applied in situations other than user specified queries.

iii. Data Modification

[0088]   An exemplary set of enhanced relational algebra mappings for data modification constructs is shown in Table 3:

Table 3

Modification	XML Algebra Operations	Enhanced Relational Mapping
INSERT	CXMLOp_Insert	CLogOp_Insert, CLogOp_UDX(re-id/validate)
DELETE	CXMLOp_Delete	CLogOp_Delete, CLogOp_UDX(validate)
REPLACE	CXMLOp_Replace	1. For value replace: CLogOp_Update 2. For tree replace: Combination of INSERT and DELETE (may be optimized)

[0089]   In addition to the operations shown in Table 3 above, other data modification constructs can be made available as XML algebra operations and converted into enhanced relational mappings. Such additional operations may include, for example, MOVE, RENAME, and UPSERT operations. The MOVE operation combines an INSERT and a DELETE. An UPSERT operation replaces an existing subtree or inserts a non-existing subtree.

[0090]   An exemplary mapping for the insertion statement "INSERT N AFTER L" is shown below:

```
INSERT (RE-ID (CROSS-PRODUCT
                    (APPLY (L, Children (Parent (L)))
                    (FLATTEN (N))))
```

[0091]   An alternative mapping is shown below:

```
INSERT (APPLY (L,
               RE-ID (UNION (Children (Parent (L),
                                       FLATTEN (N))))))
```

[0092]   The first exemplary mapping above includes the RE-ID user defined expression (UDX) above the APPLY where the RE-ID receives and processes a rowset for all L and the single subtree N.   The second exemplary mapping above includes the RE-ID under the APPLY so that FLATTEN (N) is called on all insertion locations, unlike the first plan. Thus, the first mapping is faster than the second.

[0093]   An exemplary mapping for the deletion statement "DELETE A" is shown below:

```
DELETE (FLATTEN
         (APPLY $b
           (SELECT A)                              /
           (VALIDATE
             (UNION (PROJECT 1, $b)
                    (PROJECT 0, UNNEST (Parent ($b)))))))
```

[0094]   As is apparent from the above, all or portions of the various systems, methods, and aspects of the present invention may be embodied in hardware, software, or a combination of both. When embodied in software, the methods and apparatus of the present invention, or certain aspects or portions thereof, may be embodied in the form of program code (*i.e.*, instructions). This program code may be stored on a computer-readable medium, such as a magnetic, electrical, or optical storage medium, including without limitation a floppy diskette, CD-ROM, CD-RW, DVD-ROM, DVD-RAM, magnetic tape, flash memory, hard disk drive, or any other machine-readable storage medium, wherein, when the program code is loaded into and executed by a machine, such as a computer or server, the machine becomes an apparatus for practicing the invention.   A computer on which the program code executes will generally include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device.   The program code may be

implemented in a high level procedural or object oriented programming language. Alternatively, the program code can be implemented in an assembly or machine language. In any case, the language may be a compiled or interpreted language.

[0095]   The present invention may also be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, over a network, including a local area network, a wide area network, the Internet or an intranet, or via any other form of transmission, wherein, when the program code is received and loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention.

[0096]   When implemented on a general-purpose processor, the program code may combine with the processor to provide a unique apparatus that operates analogously to specific logic circuits.

[0097]   Moreover, the invention can be implemented in connection with any computer or other client or server device, which can be deployed as part of a computer network, or in a distributed computing environment. In this regard, the present invention pertains to any computer system or environment having any number of memory or storage units, and any number of applications and processes occurring across any number of storage units or volumes, which may be used in connection with processes for persisting objects in a database store in accordance with the present invention. The present invention may apply to an environment with server computers and client computers deployed in a network environment or distributed computing environment, having remote or local storage. The present invention may also be applied to standalone computing devices, having programming language functionality, interpretation and execution capabilities for generating, receiving and transmitting information in connection with remote or local services.

[0098]   Distributed computing facilitates sharing of computer resources and services by exchange between computing devices and systems. These resources and services include, but are not limited to, the exchange of information, cache storage, and disk storage for files. Distributed computing takes advantage of network connectivity, allowing clients to leverage their collective power to benefit the entire

enterprise. In this regard, a variety of devices may have applications, objects or resources that may implicate processing performed in connection with the object persistence methods of the present invention.

[0099]   Fig. 13 provides a schematic diagram of an exemplary networked or distributed computing environment.   The distributed computing environment comprises computing objects 10a, 10b, etc. and computing objects or devices 110a, 110b, 110c, etc.   These objects may comprise programs, methods, data stores, programmable logic, etc. The objects may comprise portions of the same or different devices such as PDAs, televisions, MP3 players, personal computers, etc. Each object can communicate with another object by way of the communications network 14. This network may itself comprise other computing objects and computing devices that provide services to the system of Fig. 13, and may itself represent multiple interconnected networks.   In accordance with an aspect of the invention, each object 10a, 10b, etc. or 110a, 110b, 110c, etc. may contain an application that might make use of an API, or other object, software, firmware and/or hardware, to request use of the processes used to implement the object persistence methods of the present invention.

[0100]   It can also be appreciated that an object, such as 110c, may be hosted on another computing device 10a, 10b, etc. or 110a, 110b, etc. Thus, although the physical environment depicted may show the connected devices as computers, such illustration is merely exemplary and the physical environment may alternatively be depicted or described comprising various digital devices such as PDAs, televisions, MP3 players, etc., software objects such as interfaces, COM objects and the like.

[0101]   There are a variety of systems, components, and network configurations that support distributed computing environments. For example, computing systems may be connected together by wired or wireless systems, by local networks or widely distributed networks. Currently, many of the networks are coupled to the Internet, which provides the infrastructure for widely distributed computing and encompasses many different networks.   Any of the infrastructures may be used for exemplary communications made incident to the present invention.

[0102]   The Internet commonly refers to the collection of networks and gateways that utilize the TCP/IP suite of protocols, which are well-known in the art of computer networking. TCP/IP is an acronym for "Transmission Control Protocol/Internet Protocol." The Internet can be described as a system of geographically distributed remote computer networks interconnected by computers executing networking protocols that allow users to interact and share information over the network(s). Because of such wide-spread information sharing, remote networks such as the Internet have thus far generally evolved into an open system for which developers can design software applications for performing specialized operations or services, essentially without restriction.

[0103]   Thus, the network infrastructure enables a host of network topologies such as client/server, peer-to-peer, or hybrid architectures. The "client" is a member of a class or group that uses the services of another class or group to which it is not related. Thus, in computing, a client is a process, i.e., roughly a set of instructions or tasks, that requests a service provided by another program. The client process utilizes the requested service without having to "know" any working details about the other program or the service itself. In a client/server architecture, particularly a networked system, a client is usually a computer that accesses shared network resources provided by another computer, e.g., a server. In the example of Fig. 13, computers 110a, 110b, etc. can be thought of as clients and computer 10a, 10b, etc. can be thought of as servers, although any computer could be considered a client, a server, or both, depending on the circumstances. Any of these computing devices may be processing data in a manner that implicates the object persistence techniques of the invention.

[0104]   A server is typically a remote computer system accessible over a remote or local network, such as the Internet. The client process may be active in a first computer system, and the server process may be active in a second computer system, communicating with one another over a communications medium, thus providing distributed functionality and allowing multiple clients to take advantage of the information-gathering capabilities of the server. Any software objects utilized

pursuant to the persistence mechanism of the invention may be distributed across multiple computing devices.

[0105]   Client(s) and server(s) may communicate with one another utilizing the functionality provided by a protocol layer. For example, Hypertext Transfer Protocol (HTTP) is a common protocol that is used in conjunction with the World Wide Web (WWW), or "the Web." Typically, a computer network address such as an Internet Protocol (IP) address or other reference such as a Universal Resource Locator (URL) can be used to identify the server or client computers to each other. The network address can be referred to as a URL address. Communication can be provided over any available communications medium.

[0106]   Thus, Fig. 13 illustrates an exemplary networked or distributed environment, with a server in communication with client computers via a network/bus, in which the present invention may be employed. The network/bus 14 may be a LAN, WAN, intranet, the Internet, or some other network medium, with a number of client or remote computing devices 110a, 110b, 110c, 110d, 110e, etc., such as a portable computer, handheld computer, thin client, networked appliance, or other device, such as a VCR, TV, oven, light, heater and the like in accordance with the present invention. It is thus contemplated that the present invention may apply to any computing device in connection with which it is desirable to maintain a persisted object.

[0107]   In a network environment in which the communications network/bus 14 is the Internet, for example, the servers 10a, 10b, etc. can be servers with which the clients 110a, 110b, 110c, 110d, 110e, etc. communicate via any of a number of known protocols such as HTTP. Servers 10a, 10b, etc. may also serve as clients 110a, 110b, 110c, 110d, 110e, etc., as may be characteristic of a distributed computing environment.

[0108]   Communications may be wired or wireless, where appropriate. Client devices 110a, 110b, 110c, 110d, 110e, etc. may or may not communicate via communications network/bus 14, and may have independent communications associated therewith. For example, in the case of a TV or VCR, there may or may not be a networked aspect to the control thereof. Each client computer 110a, 110b, 110c,

110d, 110e, etc. and server computer 10a, 10b, etc. may be equipped with various application program modules or objects 135 and with connections or access to various types of storage elements or objects, across which files or data streams may be stored or to which portion(s) of files or data streams may be downloaded, transmitted or migrated. Any computer 10a, 10b, 110a, 110b, etc. may be responsible for the maintenance and updating of a database, memory, or other storage element 20 for storing data processed according to the invention. Thus, the present invention can be utilized in a computer network environment having client computers 110a, 110b, etc. that can access and interact with a computer network/bus 14 and server computers 10a, 10b, etc. that may interact with client computers 110a, 110b, etc. and other like devices, and databases 20.

[0109]   Fig. 13 and the following discussion are intended to provide a brief general description of a suitable computing device in connection with which the invention may be implemented. For example, any of the client and server computers or devices illustrated in Fig. 13 may take this form. It should be understood, however, that handheld, portable and other computing devices and computing objects of all kinds are contemplated for use in connection with the present invention, i.e., anywhere from which data may be generated, processed, received and/or transmitted in a computing environment. While a general purpose computer is described below, this is but one example, and the present invention may be implemented with a thin client having network/bus interoperability and interaction. Thus, the present invention may be implemented in an environment of networked hosted services in which very little or minimal client resources are implicated, e.g., a networked environment in which the client device serves merely as an interface to the network/bus, such as an object placed in an appliance. In essence, anywhere that data may be stored or from which data may be retrieved or transmitted to another computer is a desirable, or suitable, environment for operation of the object persistence methods of the invention.

[0110]   Although not required, the invention can be implemented via an operating system, for use by a developer of services for a device or object, and/or included within application or server software that operates in accordance with the invention. Software may be described in the general context of computer-executable

instructions, such as program modules, being executed by one or more computers, such as client workstations, servers or other devices. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Typically, the functionality of the program modules may be combined or distributed as desired in various embodiments. Moreover, the invention may be practiced with other computer system configurations and protocols. Other well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers (PCs), automated teller machines, server computers, hand-held or laptop devices, multi-processor systems, microprocessor-based systems, programmable consumer electronics, network PCs, appliances, lights, environmental control elements, minicomputers, mainframe computers and the like.

[0111]   Fig. 14 thus illustrates an example of a suitable computing system environment 100 in which the invention may be implemented, although as made clear above, the computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

[0112]   With reference to Fig. 14, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local

bus, and Peripheral Component Interconnect (PCI) bus (also known as Mezzanine bus).

[0113]   Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media include both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media include, but are not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CDROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer 110. Communication media typically embody computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and include any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media include wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

[0114]   The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and

not limitation, Fig. 14 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

[0115]   The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, Fig. 8 illustrates a hard disk drive 141 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156, such as a CD-RW, DVD-RW or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

[0116]   The drives and their associated computer storage media discussed above and illustrated in Fig. 14 provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In Fig. 14, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146 and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136 and program data 137. Operating system 144, application programs 145, other program modules 146 and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 110 through input devices such as a keyboard 162 and pointing device 161, such as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus 121, but may be connected by

other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A graphics interface 182 may also be connected to the system bus 121. One or more graphics processing units (GPUs) 184 may communicate with graphics interface 182. A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190, which may in turn communicate with video memory 186. In addition to monitor 191, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195.

[0117]   The computer 110 may operate in a networked or distributed environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in Fig. 14. The logical connections depicted in Fig. 14 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks/buses. Such networking environments are commonplace in homes, offices, enterprise-wide computer networks, intranets and the Internet.

[0118]   When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, Fig. 14 illustrates remote application programs 185 as residing on memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Conclusion

[0119]   As the foregoing illustrates, the present invention is directed to systems and methods for compiling an XML query. An extensible markup language (XML) expression included in a query is parsed to yield an abstract syntax tree. The abstract syntax tree is then transformed into a unified tree including XML algebra operations. The unified tree is then converted into a relational tree including enhanced relational algebra operations. The relational tree is then forwarded to a query processor for optimization and execution of the query by the query processor.

[0120]   While the present invention has been described in connection with the preferred embodiments of the various figures, it is to be understood that other similar embodiments may be used or modifications and additions may be made to the described embodiment for performing the same function of the present invention without deviating therefrom. Therefore, the present invention should not be limited to any single embodiment, but rather should be construed in breadth and scope in accordance with the appended claims.